BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi merupakan tahapan selanjutnya untuk merealisasikan perancanangan sebelumnya ke dalam sistem. Implementasi dilakukan dengan menerapkan rancangan pada bab sebelumnya ke dalam sebuah program menggunakan Bahasa pemrograman java yang bekerja pada komputer.

4.1.1 Implementasi Perangkat Keras

Perangkat keras berupa laptop yang digunakan untuk implementasi pembuatan program sebagai berikut :

1. Prosessor :Intel Core i3-930M CPU @1.7GHz
2. RAM RAM 4 GB (dual channel 2 GB + 2 GB)
3. Menggunakan penyimpanan yaitu HDD 500GB

4.1.2 Implementasi Perangkat Lunak

Adapun untuk perangkat lunak yang digunakan untuk menunjang implementasi memiliki spesifikasi sebagai berikut:

1. Editor Java : Neatbeans IDE versi 8.0.2
2. Kompiler Java : JDK versi 8.0 dan JRE versi 8.0
3. Sistem Operasi : Windows 10 Pro

4.1.3 Implementasi Algoritma AES Standar

Implementasi enkripsi AES terdiri dari AddRoundkey, SubBytes, ShiftRows, MixColumn. Hasil dari enkripsi akan menampilkan *chipertext* yang berupa file. Implementasi dekripsi AES terdiri dari IncSubBytes, InvShiftRows, InvMixColumn. Hasil dari dekripsi akan menampilkan *plaintext* semula. Proses implementasi dapat dilihat pada *source code* dibawah ini.

4.1.3.1 AddRoundKey

Sourecode dibawah merupakan implementasi dari persamaan 2.1 pada sub bab 2.2.1.1 yang terdiri dari parameter state, key dan object byteget. Perulangan dilakukan berdasarkan indeks baris dan kolom hingga memenuhi kondisi yang disimpan pada parameter state.

```
package AddRoundKey;
import Bitsgetter.bytegetter;
public class AddRoundKey {
    public AddRoundKey() {
    }
    publicString[][]AddRoundKey(String[][]
state,String[][] key) {
        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++)
state[i][j]=byteget.xor2hex(state[i][j],key[i][j]);
            return state;
        }
    private bytegetter byteget=new
bytegetter((byte)27);
     }
```

Gambar 4.1 Source Code AddRoundkey

Keterangan :

Parameter state : Representasi $s_{i,j}$ yang menampung *inputan file* teks

Parameter key : Representasi $k_{i,j}$ yang menampung *inputan* kunci yang digunakan oleh *user*

Object Byteget : Memanggil *method* xor2hex yaitu melakukan operasi xor pada *class bytegetter* antara 2 *statearray* berdasarkan index sesuai dengan persamaan 2.1

4.1.3.2 SubBytes

Sourecode dibawah merupakan implementasi dari persamaan 2.2 pada sub bab 2.2.1.2 yang terdiri dari 2 class yaitu *class* SBox berisikan method dengan value hexadecimal dari tabel SBox sedangkan *class* subtitue_bytes berisikan parameter cell yang memanggil method dari class sebelumnya.

```
package substitution;
import ase.SBox.SBox;
import ase.SBox.invSBOX;

public class substitute_bytes {
    public substitute_bytes()     { }
      public String  getsubstitue_bytes(String cell)
{
        return sbox.getSbox(cell);
}
     public String  invgetsubstitue_bytes(String cell)
{
        return invsbox.getinvSbox(cell);
              }
    private SBox sbox=new SBox();
    private invSBOX invsbox=new invSBOX();
    private int row;
   private int col;
}
```

```
package ase.SBox;

import Hexdecimal_converter.decimalhex;
public class SBox {
    public SBox() {
    }
    public String getSbox(String cell)
    {
          row=decimalhex.getDecimal(cell.charAt(0));
          col=decimalhex.getDecimal(cell.charAt(1));
        return sbox[row][col];
    }
    private  String sbox[][]=new String[][] :

    private decimalhex decimalhex=new decimalhex();
    private int row;
    private int col;
}
```

Gambar 4.2 Source Code Subbytes

Keterangan :

Class substitute_bytes

Parameter cell : representasi dari $s_{i,j}$ yaitu *state array* yang akan melakukan proses substitusi dengan lookup table s-box.

Objek sbox : memanggil method getSbox pada class SBox yang merepresentasikan proses subtitusi dari $b_{i,j}$

Class SBox

Variabel row : mendapatkan value cell menjadi dua karakter dengan index ke-0 menjadi value

baris dengan lookuptable s- box. Variabel ini mereperesntasikan indeks ke-i pada suku $s_{i,j}$ dan $b_{i,j}$ pada persamaan 2.2

variabel col : mendapatkan value cell menjadi dua karakter index ke-1 menjadi value kolom dengan lookuptable s- box. Variabel ini mereperesntasikan indeks ke-j pada suku $s_{i,j}$ dan $b_{i,j}$ pada persamaan 2.2.

array sbox : *array* dua dimensi yang memiliki *value* dari *lookup table* s-box berdasarkan gambar 2.2.

4.1.3.3 ShiftRow

Sourecode dibawah merupakan implementasi dari persamaan 2.3 yang akan melakukan proses permutasi sehingga menghasilkan matriks 4x4 berdasarkan gambar 2.3 pada sub bab 2.2.1.3. Class berisikan method shiftrows dan rowshift yang melakukan pergeseran berdasarkan indeks baris.

```
package Shiftrow;

public class shiftrows {

    public shiftrows() {
    }
    public String[][] shiftrows(String state[][])
    {
       for(int i=1;i<4;i++)
       {
           state=rowshift(i,state);
       }
        return state;
    }
```

```
    public String[][] rowshift(int row,String[][]
state)
    {
         String temp;
         for(int num=0;num<row;num++)
         {
        temp=state[row][0];
         for(int i=0;i<3;i++)
              {
                  state[row][i]=state[row][i+1];
              }
               state[row][3]=temp;
               }
               return state;
          }
}
```

Gambar 4.3 Source Code ShiftRows

Keterangan :

Parameter *state* : *input file plaintext* dan akan menampung nilai sementara hasil proses setiap transformasi

Method shiftrows : memanggil proses pada method rowshift dengan parameter state dan paramteer i untuk melakukan perulangan sebanyak n sesuai pada persamaan 2.3 yang kemudian ditampung pada variabel state

Method rowshift : Memiliki dua parameter row dan state dengan melakukan proses permutasi value index word. Method ini merupakan representasi dari persamaan 2.3.

4.1.3.4 MixColumn

Sourecode dibawah merupakan implementasi dari ilustrasi pada Gambar 2.4 yang melakukan proses perkalian matriks sesuai

dengan persamaan 2.4. Class terdiri dari beberapa variable yang menampung consntan matriks maupun hasil perkalian xor pada proses mixcolumn.

```
package Mixcolumns;
import Bitsgetter.bytegetter;
import ase.arrayprinter;
public class mixcolumns {
    public mixcolumns() {
    }
    public String[][]  mixcolumns(String[][] state) {
        bytegetter=new bytegetter((byte)27);
        aftermix=new String[4][4];
        data=new byte[4];

        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++) {
            for(int k=0;k<4;k++)
            {

data[k]=bytegetter.Xorbytegetter(trans[i][k],state[k][j]);
            }
            aftermix[i][j]=bytegetter.XorAll(data);
            }
  return aftermix;
         }
    String[][] trans=new String[][]
    {{"02","03","01","01"},
     {"01","02","03","01"},
     {"01","01","02","03"},
     {"03","01","01","02"},
    };
```

```
    private bytegetter bytegetter;
    private String[][] aftermix=new String[4][4];
    private byte[] data=new byte[4];
}
```

Gambar 4.4 Source Code MixColumn

Keterangan :

Parameter state	: menampung nilai sementara dari hasil proses transformasi sebelumnya yaitu representasi dari $s_{i,j}$
Variabel trans	: menampung matriks konstan yang digunakan pada proses aftermix berdasarkan ilustrasi mixcolumn pada gambar 2.4.
Variabel data	: menampung nilai sementara hasil xor antara word dan kontan matriks sesuai pada gambar 2.4.
Variable aftermix	: menampung hasil akhir dari proses xor seluruh indeks dari variable data

4.1.3.5 Key Expansion

Implementasi dibawah dilakukan berdasarkan beberapa persaman yang terdapat dalam subbab 2.2.1.5. Class terdiri dari method subword, rotword , rcxor dan keyexpansion yang melakukan proses secara keseluruhan berdasarkan jumlah kunci yang akan digunakan untuk setiap putaran.

```
package AddRoundKey;
public class KeyExpansion {
public KeyExpansion() {
   }
    public void keyExpansion(String[][] mykey){
        words[0]=new String[4];
```

```
        words[1]=new String[4];
        words[2]=new String[4];
        words[3]=new String[4];
        words[0]=mykey[0];
        words[1]=mykey[1];
        words[2]=mykey[2];
        words[3]=mykey[3];
         for(int i=4;i<44;i++) {
            temp=words[i-1];
                  System.out.println(temp[0]);
                  if(i%4==0){
                      temp=
      RCxor((SubWord(Rotword(temp))),i/4);
            }
        words[i]=xor2words(words[i-4],temp);
                          }
        arrayprinter.printarray(words,"key
Expantion");    }
    public String[] Rotword(String[] word) {
    String[] afterRot=new String[4];
          for(int i=0;i<3;i++)
             afterRot[i]=word[i+1];
        afterRot[3]= word[0];
        return afterRot;
         }
   public String[] SubWord(String[] word) {
        String [] aftersub=new String[4];
        for(int i=0;i<4;i++) {
          aftersub[i]=sbox.getSbox(word[i])}
        return aftersub;
    }
    public String[] RCxor(String[] word,int i) {
        RCword[0]=RCtable[i-1]
        String[] w= xor2words(RCword,word);
```

```
            return w;
          }

    private  SBox sbox=new SBox();

    private          bytegetter          byteget=new
bytegetter((byte)27);

    String [] temp=new String[4];

   String[]RCtable=newString[]
{"01","02","04","08","10","20","40","80","1B","36"};

    String     []     RCword=new           String[]
{"00","00","00","00",};

    String [] word1=new String[4];

    String [] word2=new String[4];

    String [] word3=new String[4];

    String [] word4=new String[4];

    String [][] words=new String[44][4];

    String [] newWord=new String[4];

    String k;  }
```

Gambar 4.5 Key Expansion

Keterangan :

Method Rotword : melakukan rotasi sesuai dengan indeks ke-3 pada word keluaran sesuai dengan persamaan 2.5 dengan w_{i-1}

Method SubWord : melakukan subtitusi menggunakan tabel s-box pada gambar 2.2 dengan word keluaran atau w_{i-1} sesuai dengan persamaan 2.6

Method RCxor : fungsi yang melakukan xor antar word dan konstan RCon yang digunakan

Method KeyExpansion : fungsi yang memanggil method sebelumnya berdasarkan persamaan 2.7 sesuai dengan perulangan key schedule dengan melakukan xor antar word sesuai indeks

4.1.3.6 InvSubBytes

Implementsi dilakukan berdasarkan persamaan 2.9 yang terdapat pada subbab 2.2.2.2. Class berisikan value hexadecimal dari tabel inverse sbox dengan parameter cell.

```
      package ase.SBox;
      import Hexdecimal_converter.decimalhex;


      public class invSBOX {
          public invSBOX() {
          }
          public String getinvSbox(String cell) {
              row=decimalhex.getDecimal(cell.charAt(0));
              col=decimalhex.getDecimal(cell.charAt(1));
              return invsbox[row][col];
          }
          private  String invsbox[][]=new String[][]{
          private decimalhex decimalhex=new
      decimalhex();
          private int row;
          private int col;
}
```

Gambar 4.6 Source Code Invsubbytes

Keterangan :

Method getinvSbox : mengambil indeks hexadecimal dari parameter cell kemudian inisialisasi menjadi baris dan kolom

Variabel invsbox : array dua dimensi yang berisikan baris dan kolom berupa bilangan hexadecimal dari lookup tabel invsbox

Variabel row : value cell menjadi dua karakter indeks ke-0 yang menjadi value baris pada proses subtitusi menggunakan tabel invsbox

Variabel col : value cell menjadi karakter indeks ke-1 yang menjadi value kolom pada proses subtitusi menggunakan tabel invsbox

Parameter cell : value index state array yang akan dilakukan proses subtitusi dengan menggunakan lookup table invsbox

4.1.3.7 InvShiftRow

Implementasi dilakukan berdasarkan persamaan 2.10. Class berisikan method invshiftrows dan rowshift yang melakukan pergeseran sesuai dengan indeks baris dengan parameter state array 2 dimensi.

```
package Shiftrow;
import ase.arrayprinter;
public class invshiftrows {
 public invshiftrows() {
    }
    public String[][] invshiftrows(String state[][])
    {
       for(int i=1;i<4;i++)
       {          state=rowshift(i,state);       }
            return state;
        }
 public String[][] rowshift(int row,String[][] state)
    {
         String temp;
         for(int num=0;num<row;num++)
         {
        temp=state[row][3];
        for(int i=3;i>0;i--)
        {
            state[row][i]=state[row][i-1];
        }
         state[row][0]=temp;     }
         return state;    }}
```

Gambar 4.7 Source Code InvShiftRows

Keterangan :

Parameter state	: menampung nilai stateplain dari proses transformasi sebelumnya
Variabel row	: variable yang menampung index pertama dari array dua dimensi yang merepresentasikan baris
Method invshiftrows	: menampung nilai dari variable state yang berisikan perulangan untuk melakukan proses pergeseran sesuai dengan baris atau *i* yang kemudian diproses pada method rowshift
Method rowshift	: method yang berisikan proses permutasi word berdasarkan indeks value sesuai dengan ilustrasi pada gambar 2.6.

4.1.3.8 InvMixColumn

Sourcecode dibawah merupakan implementasi dari persamaan 2.11 yang terdiri dari beberapa variable yang menampung nilai konstan inverse matriks dan hasil xor dari state.

```
package Mixcolumns;
import Bitsgetter.bytegetter;
public class invmixcolumns {
    public invmixcolumns() {
    }
    public String[][]  invmixcolumns(String[][] state) {
        bytegetter=new bytegetter((byte)27);
        aftermix=new String[4][4];
        data=new byte[4];
         for(int i=0;i<4;i++)
            for(int j=0;j<4;j++) {
              for(int k=0;k<4;k++)
```

```
data[k]=bytegetter.Xorbytegetter(trans[i][k],state[k][j]);
                    aftermix[i][j]=bytegetter.XorAll(data);
                    }
                return aftermix;
            }
    String[][] trans=new String[][]
    {{"0E","0B","0D","09"},
     {"09","0E","0B","0D"},
     {"0D","09","0E","0B"},
     {"0B","0D","09","0E"},
    };
    private bytegetter bytegetter ;
    private String[][] aftermix=new String[4][4];
    private byte[] data=new byte[4];
}
```

Gambar 4.8 Source Code InvMixColumns

Keterangan :

Parameter *state* : menampung nilai sementara hasil proses dari proses transformasi sebelumnya

Variabel aftermix : menampung hasil akhir xor indeks dari proses invmixcolumn

Variabel trans : menampung value konstan matriks yang digunakan pada invmixcolumn

Variabel data : menampung value sementara dari proses xor antara stateplain dan konstan matriks

Method xorbytegetter : method yang memiliki fungsi melakukan proses xor antar dua variable array kemudian di xor kan sesama index value masing-masing variable

Method xorall : method yang melakukan proses seluruh hasil dari proses xorbytegetter kemudian

ditampung pada variable aftermix sesuai pada persamaan 2.11.

4.1.4 Implementasi Algoritma AES Modifikasi

Implementasi AES Modifikasi dilakukan berdasarkan penambahan matriks Polybius. Matriks yang digunakan yaitu 6x6 dan 10x10. Modifikasi dilakukan secara berturut-turut pada perubahan plainteks, kunci dan plainteks.

4.1.4.1 Matriks Polybius 6x6

Sourcecode dilakukan berdasarkan persamaan 2.12 yang terdiri dari beberapa variable dan method yang akan melakukan proses subtitusi menggunakan matriks 6x6.

```
public class PolybiousSquare6x6 {
private final char[][] matrix = {
                {'A', 'B', 'C', 'D', 'E', 'F'},
                {'G', 'H', 'I', 'J', 'K', 'L'},
                {'M', 'N', 'O', 'P', 'Q', 'R'},
                {'S', 'T', 'U', 'V', 'W', 'X'},
                {'Y', 'Z', '1', '2', '3', '4'},
                {'5', '6', '7', '8', '9', ' '}};
            public String encrypt(String plain) {
                plain = plain.toUpperCase();
                StringBuilder result = new StringBuilder();
                for (int i = 0; i < plain.length(); i++) {

      result.append(findChar(plain.charAt(i)));
                }
                return result.toString();
            }

    public String decrypt(String chiper) {

        StringBuilder result = new StringBuilder();

        for (int i = 0; i < chiper.length(); i += 2) {

            introw=
Integer.parseInt(Character.toString(chiper.charAt(i)));

   intcol=
Integer.parseInt(Character.toString(chiper.charAt(i+1)));

                    result.append(matrix[row][col]);

        }
```

```
        return result.toString();
    }
    private String findChar(char input) {
        String result = "";
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (input == matrix[i][j]) {
                    result = String.format("%d%d", i, j);
                    break;
                }
            }
        }
        return result;
    }
```

Gambar 4.9 Source Code Polybius 6x6

Keterangan :

Parameter *plain*	: menampung value plain sementara yang akan diproses pada tahap selanjutnya
Method findchar	: mencari indeks pada setiap karakter yang digunakan pada matriks berdasarkan baris atau *i* dan kolom atau *j* sesuai dengan persamaan 2.12
Parameter chipper	: menampung value chiper berdasarkan row dan col yang digunakan pada matriks untuk mendapatkan char sesuai dengan indeks pada matriks
Variabel row	: mencari indeks pertama sebagai baris pada matriks
Variabel col	: mencari indeks kedua sebagai kolom pada matriks

Variabel matrix : menyimpan value dari matriks 6x6 polybius sesuai dengan gambar 2.9.

4.1.4.2 Matriks Polybius 10x10

Sourcecode dilakukan berdasarkan persamaan 2.12 yang terdiri dari beberapa variable dan method yang akan melakukan proses subtitusi menggunakan matriks 10x10.

```
public class PolybiousSquare10x10 {


    private final char[][] matrix = {
        {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j'},
        {'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't'},
        {'u', 'v', 'w', 'x', 'y', 'z', '0', '1',
'2', '3'},
        {'4', '5', '6', '7', '8', '9', ' ', '!',
'@', '#'},
        {'$', '%', '^', '&', '*', '(', ')', '-',
'+', '{'},
        {'}', '_', '=', '[', ']', ';', '\"', '.',
',', '?'},
        {'<', '>', ':', '\'', '/', '\\', 'α', 'β',
'γ', 'δ'},
        {'ε', 'ν', 'η', 'θ', 'λ', 'μ', 'π', 'σ',
'φ', '∞'},
        {'ʃ', 'υ', 'ʒ', '÷', '≠', '¥', '≤', '≥',
'↓', '→'},
        {'←', '↑', '↔', '¶', 'Σ', 'Ω', '|', 'ω',
'Δ', '~'}};


    public String encrypt(String plain) {


        plain = plain.toLowerCase();


        StringBuilder result = new StringBuilder();
```

```
        for (int i = 0; i < plain.length(); i++) {

result.append(findChar(plain.charAt(i)));
              }
        return result.toString();
    }


    public String decrypt(String chiper) {
        StringBuilder result = new StringBuilder();


        for (int i = 0; i < chiper.length(); i +=
2) {
            int row =
Integer.parseInt(Character.toString(chiper.charAt(i
)));
            int col =
Integer.parseInt(Character.toString(chiper.charAt(i
+ 1)));
                result.append(matrix[row][col]);
            }
            return result.toString();
        }
    private String findChar(char input) {
        String result = "";
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length;
j++) {
                if (input == matrix[i][j]) {
                    result = String.format("%d%d",
i, j);
                    break;
                }
            }
        }
        return result;    }
```

Gambar 4.10 Source Code Polybius 10x10

Keterangan :

Parameter plain	: menampung nilai sementara hasil proses sebelumnya
Method findchar	: mencari indeks pada setiap karakter sesuai dengan perulangan input karakter yang digunakan sesuai dengan persamaan 2.13
Parameter chipper	: menampung value chiper berdasarkan row dan col yang digunakan pada matriks untuk mendapatkan char sesuai dengan indeks pada matriks
Variabel row	: mencari karakter pertama sebagai baris berdasarkan matriks
Variabel col	: mencari karakter kedua sebagai kolom berdasarkan matriks
Variabel matrix	: menyimpan value dari matriks 10x10 polybius sesuai dengan gambar 2.10

4.1.4.1 AES Polybius 6x6 Plain

Implementasi dilakukan berdasarkan ilsutrasi gambar 3.3 yang berisikan beberapa variable dan method untuk melakukan subtitusi plainteks yang digunakan berdasarkan matriks 6x6.

```
import polybioussquare.PolybiousSquare6x6;
import ui.UIListener;
public class AESPolybius6Plain {

    private AESAlgorithm aes;
    private PolybiousSquare6x6 poli;
    private String key;

    private UIListener listener;
    public void setListener(UIListener listener)
{
        this.listener = listener;
    }
    public AESPolybius6Plain(String key) {
        this.key = key;
```

```
            aes = new AESAlgorithm(key);
            poli = new PolybiousSquare6x6();
        }
        public String encrypt(String plain) {
            if (key.length() != 16 && listener !=
    null) {
                listener.insertLog("ERROR : Kunci
    tidak valid!");
                listener.showMessage("Panjang kunci
    harus 16 Karakter");
                return "";
            }
            String matrik = poli.encrypt(plain);
            if (listener != null) {
                listener.insertLog("Matrix
    polybius:\n" + matrik);
            }

            String result = aes.encrypt(matrik);
            if (listener != null) {
                listener.insertLog("Chiper:\n" +
    result);
            }
            return result;
        }
        public String decrypt(String chiper) {
            if (key.length() != 16 && listener !=
    null) {
                listener.insertLog("ERROR : Kunci
    tidak valid!");
                listener.showMessage("Panjang kunci
    harus 16 Karakter");
                return "";
            }
            String matrik = aes.decrypt(chiper);
            if (listener != null) {
                listener.insertLog("Matrix
    polybius:\n" + matrik);
            }
            String result = poli.decrypt(matrik);
            if (listener != null) {
                listener.insertLog("Plain:\n" +
    result);
            }
            return result;
    }
```

Gambar 4.11 Source Code Polybius 6x6 Plain

Keterangan :

Parameter *key*	: menampung nilai dari *key* yang digunakan untuk proses enkripsi dan dekripsi
Object poli	: menampung value matriks Polybius 6x6 yang telah dibuat pada class sebelumnya
Method encypt	: berisikan perulangan apabila *key* tidak memenuhi 16 karakter maka proses selanjutnya tak berjalan
Variabel matrik	: convert plain ke matriks yang digunakan yaitu 6x6 kemudian melakukan proses enkripsi pada AES
Method decrypt	: method yang memiliki fungsi mengembalikan plain dengan melakukan fungsi dekripsi pada AES kemudian convert pada matriks poli 6x6 dengan parameter chipper

4.1.4.3 AES Polybius 6x6 Plainkey

Implementasi dilakukan berdasarkan ilsutrasi gambar 3.7 yang berisikan beberapa variable dan method untuk melakukan subtitusi plainteks dank unci yang digunakan berdasarkan matriks 6x6.

```
import polybioussquare.PolybiousSquare6x6;
import ui.UIListener;
    public class AESPolybius6PlainKey {
    private AESAlgorithm aes;
    private PolybiousSquare6x6 poli;
    private String key;
    private UIListener listener;
    public void setListener(UIListener listener) {
        this.listener = listener;
          }
  public AESPolybius6PlainKey(String key) {
```

```
    poli = new PolybiousSquare6x6();
     this.key = key;
          }
    public String encrypt(String plain) {
        if (key.length() != 8 && listener != null) {
        listener.insertLog("ERROR : Kunci tidak
      valid!");
        listener.showMessage("Panjang kunci harus 8
      Karakter");
            return "";
        }
        String matricKey = poli.encrypt(key);
        if (listener != null) {
            listener.insertLog("Key to Matrix
polybius:\n" + matricKey);
        listener.insertLog("Using Matrix Key as AES
key");
        }
         aes = new AESAlgorithm(matricKey);
         String matrik = poli.encrypt(plain);
        if (listener != null) {
            listener.insertLog("Matrix polybius:\n"
+ matrik);
        }
        String result = aes.encrypt(matrik);
        if (listener != null) {
            listener.insertLog("Chiper:\n"        +
result);
        }
    return result; }
    public String decrypt(String chiper) {
        if (key.length() != 8 && listener != null) {
            listener.insertLog("ERROR : Kunci tidak
valid!");
```

```
            listener.showMessage("Panjang      kunci
harus 8 Karakter");
            return "";
              }
         String matricKey = poli.encrypt(key);
           if (listener != null) {
                  listener.insertLog("Key  to  Matrix
      polybius:\n" + matricKey);
            listener.insertLog("Using Matrix Key as
AES key");
        }
        aes = new AESAlgorithm(matricKey);
        String matrik = aes.decrypt(chiper);
        if (listener != null) {
            listener.insertLog("Matrix  polybius:\n"
+ matrik);
            }
       String result = poli.decrypt(matrik);
        if (listener != null) {
            listener.insertLog("Plain:\n" + result);
        }
        return result;   }
```

Gambar 4.12 Source Code Polybius 6x6 PlainKey

Keterangan :

Parameter *key*	: menampung nilai dari *key* yang digunakan untuk proses enkripsi dan dekripsi
Object poli	: menampung value matriks Polybius 6x6 yang telah dibuat pada class sebelumnya
Variabel matrickey	:menampung hasil convert key menggunakan matriks Polybius 6x6
Variabel matrik	: convert plain ke matriks yang digunakan yaitu 6x6 kemudian melakukan proses enkripsi pada AES

Method decrypt : method yang memiliki fungsi mengembalikan plain dan key dengan melakukan fungsi dekripsi pada AES kemudian convert pada matriks poli 6x6 untuk masing-masing *key* dan plain dengan parameter matriks

4.1.4.4 AES Polybius 10x10 Plain

Implementasi dilakukan berdasarkan ilsutrasi gambar 3.3 yang berisikan beberapa variable dan method untuk melakukan subtitusi plainteks yang digunakan berdasarkan matriks 10x10.

```
import polybioussquare.PolybiousSquare10x10;
import ui.UIListener;
public class AESPolybius10Plain {
    private AESAlgorithm aes;
    private PolybiousSquare10x10 poli;
    private String key;
    private UIListener listener;
    public void setListener(UIListener listener) {
        this.listener = listener;
    }
    public AESPolybius10Plain(String key) {
        this.key = key;
        aes = new AESAlgorithm(key);
        poli = new PolybiousSquare10x10();
    }
    public String encrypt(String plain) {
        if (key.length() != 16 && listener != null)
{
            listener.insertLog("ERROR : Kunci tidak
valid!");
            listener.showMessage("Panjang kunci
harus 16 Karakter");
```

```
            return "";
        }
        String matrik = poli.encrypt(plain);
        if (listener != null) {
            listener.insertLog("Matrix polybius:\n"
+ matrik);
              }
        String result = aes.encrypt(matrik);
        if (listener != null) {
            listener.insertLog("Chiper:\n" +
result);
              }
        return result;
         }
    public String decrypt(String chiper) {
        if (key.length() != 16 && listener != null)
{
            listener.insertLog("ERROR : Kunci tidak
valid!");
            listener.showMessage("Panjang kunci
harus 16 Karakter");
            return "";
        }
        String matrik = aes.decrypt(chiper);
        if (listener != null) {
            listener.insertLog("Matrix polybius:\n"
+ matrik);
        }
        String result = poli.decrypt(matrik);
        if (listener != null) {
            listener.insertLog("Plain:\n" +
result);
        }
        return result;
```

```
    }
```

Gambar 4.13 Source Code Polybius 10x10 Plain

Keterangan :

Parameter *key*	: menampung nilai dari *key* yang digunakan untuk proses enkripsi dan dekripsi
Object poli	: menampung value matriks Polybius 10x10
Variabel matrik	: convert plain ke matriks yang digunakan yaitu 10x10 kemudian melakukan proses enkripsi pada AES
Method decrypt	: method yang memiliki fungsi mengembalikan plain dengan melakukan fungsi dekripsi pada AES kemudian convert pada matriks poli 10x10 dengan parameter chipper

4.1.4.5 AES Polybius 10x10 Plainkey

Implementasi dilakukan berdasarkan ilsutrasi gambar 3.7 yang berisikan beberapa variable dan method untuk melakukan subtitusi plainteks dank unci yang digunakan berdasarkan matriks 10x10.

```
import polybioussquare.PolybiousSquare10x10;
import ui.UIListener;
public class AESPolybius10PlainKey {
    private AESAlgorithm aes;
    private PolybiousSquare10x10 poli;
    private String key;
    private UIListener listener;
    public void setListener(UIListener listener) {
        this.listener = listener;
```

```
    }
    public AESPolybius10PlainKey(String key) {
        poli = new PolybiousSquare10x10();
        this.key = key;
    }
    public String encrypt(String plain) {
        if (key.length() != 8 && listener != null) {
            listener.insertLog("ERROR : Kunci tidak
valid!");
            listener.showMessage("Panjang kunci harus
8 Karakter");
            return "";
        }
        String matricKey = poli.encrypt(key);
        if (listener != null) {
            listener.insertLog("Key to Matrix
polybius:\n" + matricKey);
            listener.insertLog("Using Matrix Key as AES
key");
        }
        aes = new AESAlgorithm(matricKey);
       String matrik = poli.encrypt(plain);
        if (listener != null) {
            listener.insertLog("Matrix polybius:\n" +
matrik);
        }
        String result = aes.encrypt(matrik);
        if (listener != null) {
            listener.insertLog("Chiper:\n" + result);
        }
        return result;
    }
    public String decrypt(String chiper) {
        if (key.length() != 8 && listener != null) {
```

```
            listener.insertLog("ERROR : Kunci tidak
valid!");
            listener.showMessage("Panjang kunci harus
8 Karakter");
            return "";
        }
        String matricKey = poli.encrypt(key);
        if (listener != null) {
            listener.insertLog("Key to Matrix
polybius:\n" + matricKey);
            listener.insertLog("Using Matrix Key as AES
key");
        }
        aes = new AESAlgorithm(matricKey);
        String matrik = aes.decrypt(chiper);
        if (listener != null) {
            listener.insertLog("Matrix polybius:\n" +
matrik);
        }
        String result = poli.decrypt(matrik);
        if (listener != null) {
            listener.insertLog("Plain:\n" + result);
        }
        return result; }
```

Gambar 4.14 Source Code Polybius 10x10 PlainKey

Keterangan :

Parameter *key*	: menampung nilai dari *key* yang digunakan untuk proses enkripsi dan dekripsi
Object poli	: menampung value matriks Polybius 10x10
Variabel matrickey	:menampung value hasil convert key menggunakan matriks Polybius 10x10

Variabel matrik	: convert plain ke matriks poli kemudian melakukan proses transformasi enkripsi pada AES
Method decrypt	: method yang memiliki fungsi mengembalikan plain dan key dengan melakukan fungsi dekripsi pada AES kemudian convert pada matriks poli 10x10 untuk masing-masing *key* dan plain

4.2 Pengujian

Pengujian dilakukan berdasarkan hasil implementasi pada sub bab sebelumnya. Pengujian yang dilakukan berdasarkan tingkat keamanan algoritma dan performa waktu. Pengujian keamanan yang digunakan berdasarkan perubahan bit menggunakan *Avalanche Effect* , *chi square* dan pengujian performa waktu enkrispi maupun dekripsi.

4.2.1 Pengujian Avalanche Effect

Pengujian AE dilakukan berdasarkan rumus pada subab 2.8 kemudian didapatkan hasil untuk masing-masing algoritma sesuai dengan tabel dibawah ini.

Tabel 4.1 Hasil Pengujian AE AES Standar

Size File	Perubahan bit	Avalanche effect(%)
11.003 *bytes*	129	43%
11.993 *bytes*	425	45%
12.598 *bytes*	1935	45%
13.827 *bytes*	2937	49%
14.587 *bytes*	3455	49%
Rata-rata		46,2%

Berdasarkan tabel 4.1 hasil pengujian *avalanche effect* didapatkan rata-rata secara keseluruhan dari masing-masing ukuran *file* yaitu 46,2%. Hasil untuk setiap ukuran didapatkan persentase bervariasi dikarenakan plainteks yang digunakan bervariasi dan memiliki karakter yang berbeda sehingga plainteks mempengaruhi tingkat efisiensi pengujian menggunakan *avalanche effect*.

Tabel 4.2 Hasil Pengujian AE Modifikasi AES 1 pada Plain Matriks 6x6

Size File	Perubahan bit	Avalanche effect(%)
11.003 *bytes*	152	52%
11.993 *bytes*	498	52%
12.598 *bytes*	2074	48%
13.827 *bytes*	3112	52%
14.587 *bytes*	3854	55%
Rata-rata		51,2%

Berdasarkan tabel 4.2 hasil pengujian *avalanche effect* didapatkan rata-rata secara keseluruhan dari masing-masing ukuran *file* yaitu 51,2%. Hasil modifikasi mempengaruhi tingkat *avalanche effect* dikarenakan penambahan matriks yang digunakan dalam perubahan plainteks.

Tabel 4.3 Hasil Pengujian AE Modifikasi AES 1 pada Plain Matriks 10x10

Size File	Perubahan bit	Avalanche effect(%)
11.003 *bytes*	140	46%
11.993 *bytes*	533	55%
12.598 *bytes*	2150	50%

13.827 *bytes*	2899	49%
14.587 *bytes*	3763	53%
Rata-rata		50,6%

Berdasarkan tabel 4.3 hasil pengujian *avalanche effect* didapatkan rata-rata secara keseluruhan dari masing-masing ukuran *file* yaitu 50,6%. Hasil modifikasi penggunaan matriks 10x10 mempengaruhi tingkat *avalanche effect*. Matriks yang digunakan dalam modifikasi mempengaruhi tingkat *avalanche effect* dapat disimpulkan dari rata-rata pengujian untuk masing-masing modifikasi.

Tabel 4.4 Hasil Pengujian AE Modifikasi AES 2 pada Plain dan Key Matriks 6x6

Size File	Perubahan bit	Avalanche effect(%)
11.003 *bytes*	154	51%
11.993 *bytes*	442	46%
12.598 *bytes*	2380	55%
13.827 *bytes*	3010	51%
14.587 *bytes*	3920	56%
Rata-rata		51,8%

Berdasarkan tabel 4.4 hasil pengujian *avalanche effect* didapatkan rata-rata secara keseluruhan dari masing-masing ukuran *file* yaitu 51,8%. Hasil modifikasi mempengaruhi tingkat *avalanche effect* dikarenakan proses modifikasi yang dilakukan pada plainteks dan kunci.

Tabel 4.5 Hasil Pengujian AE Modifikasi AES 2 pada Plain dan Key Matriks 10x10

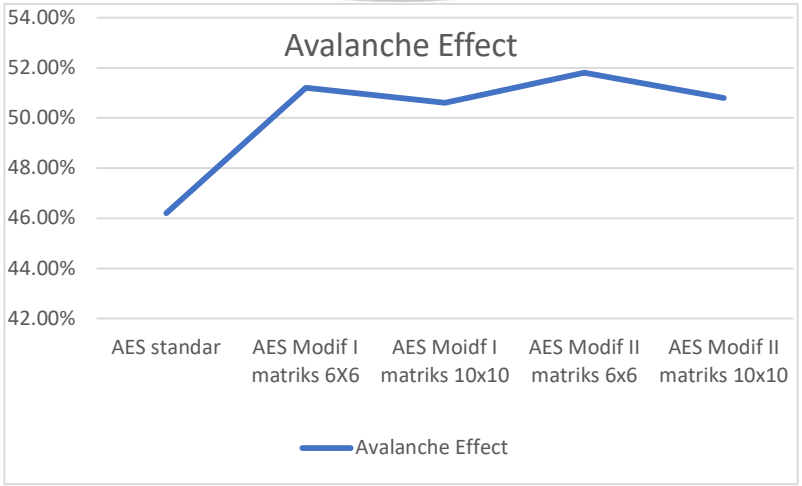
Size File	Perubahan bit	Avalanche effect(%)
11.003 *bytes*	135	45%
11.993 *bytes*	472	49%
12.598 *bytes*	2364	55%
13.827 *bytes*	2970	50%
14.587 *bytes*	3871	55%
Rata-rata		50,8%

Berdasarkan tabel 4.4 hasil pengujian *avalanche effect* didapatkan rata-rata secara keseluruhan dari masing-masing ukuran *file* yaitu 50,8%. Hasil modifikasi mempengaruhi tingkat *avalanche effect* dikarenakan proses modifikasi yang dilakukan pada perubahan plainteks dan kunci.

4.2.2 Analisa Hasil Pengujian Avalanche Effect

Hasil pengujian yang didapatkan memiliki tingkat persentase *avalanche effect* yang berbeda sehingga dapat disimpulkan sesuai dengan grafik berikut

Grafik 4.1 Hasil Avalanche Effect



Berdasarkan grafik 4.1 persentase tertinggi terdapat pada AES modifikasi II pada plain dan *key* sesuai pada point 1 subab 3.4.1 yaitu algoritma dapat dikatakan aman karena memiliki rata-rata persentase yang lebih tinggi sedangkan persentase terendah yaitu AES standar sesuai pada point 2 bahwa algoritma dikatakan kurang aman.

4.2.3 Pengujian Chi Square

Pengujian Chi Square dilakukan berdasarkan scenario pengujian pada sub bab 3.3.1. Hasil dari pengujian akan mendapatkan satu dari dua kesimpulan yaitu "H0 diterima dan H1 ditolak" atau "H0 ditolak dan H1 diterima".

Tabel 4.6 Hasil Pengujian Chi Square

No	Parameter Uji	Taraf Nyata	Derajat Kebebasan	Chi Square Table	Chi Square Hitung	Hasil
1	*Avalanche Effect* Modifikasi 6x6 pada Plain	0,05	4	9,4877	0,5859	H0 diterima, H1 ditolak
2	*Avalanche Effect* Modifikasi 10X10 pada Plain	0,05	4	9,4877	1,0647	H0 diterima, H1 ditolak
3	*Avalanche Effect* Modifikasi 6x6 pada Plain dan Key	0,05	4	9,4877	1,3407	H0 diterima, H1 ditolak
4	*Avalanche Effect* Modifikasi 10x10 pada Plain dan Key	0,05	4	9,4877	1,022	H0 diterima, H1 ditolak

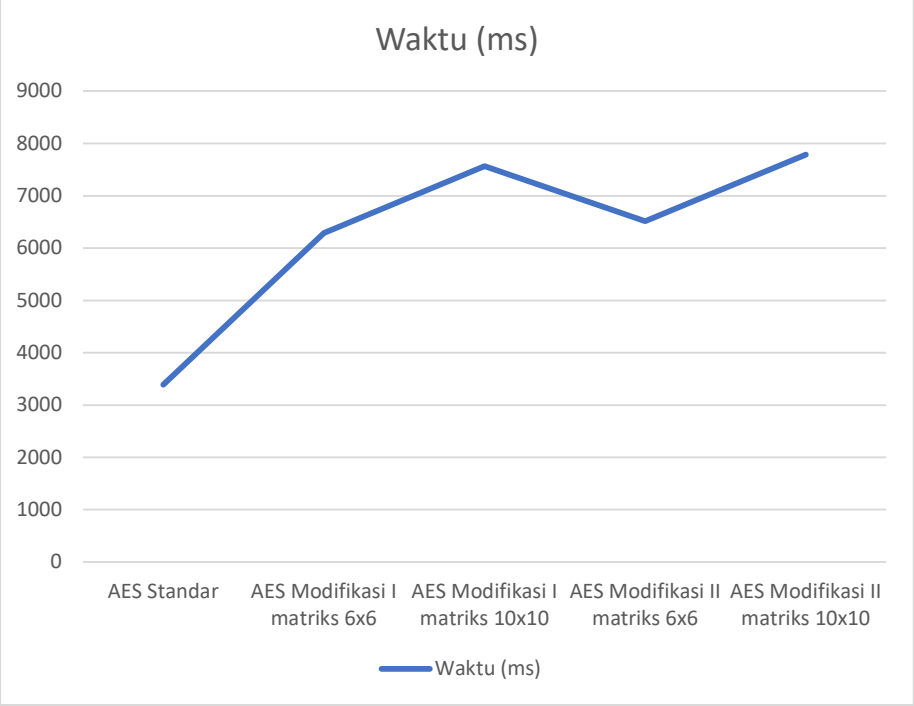
Tabel 4.6 menunjukkan hasil pengujian Chi Square dengan taraf nyata 0,05 dan derajat kebebasan 4. Nilai dari *chi square table* dilihat dari

tabel *chi square* berdasarkan taraf nyata yang digunakan.Jika Chi Square hitung lebih kecil dari Chi Square table maka H0 diterima dan H1 ditolak. Penelitian Chi Square menghasilkan kesimpulan sesuai pada point 1 subbab 3.4.2 bahwa setiap modifikasi menggunakan *Polybius* dapat meningkatkan Avalanche Effect sebesar 5%. Hasil didapatkan berdasarkan taraf nyata dan derajat kebebasan yang telah disebutkan sebelumnya.

4.2.4 Pengujian Performa Waktu

Pengujian waktu dilakukan berdasarkan scenario pengujian pada sub bab 3.3.2. Hasil pengujian dapat dilihat dari grafik dibawah ini untuk masing-masing algoritma.

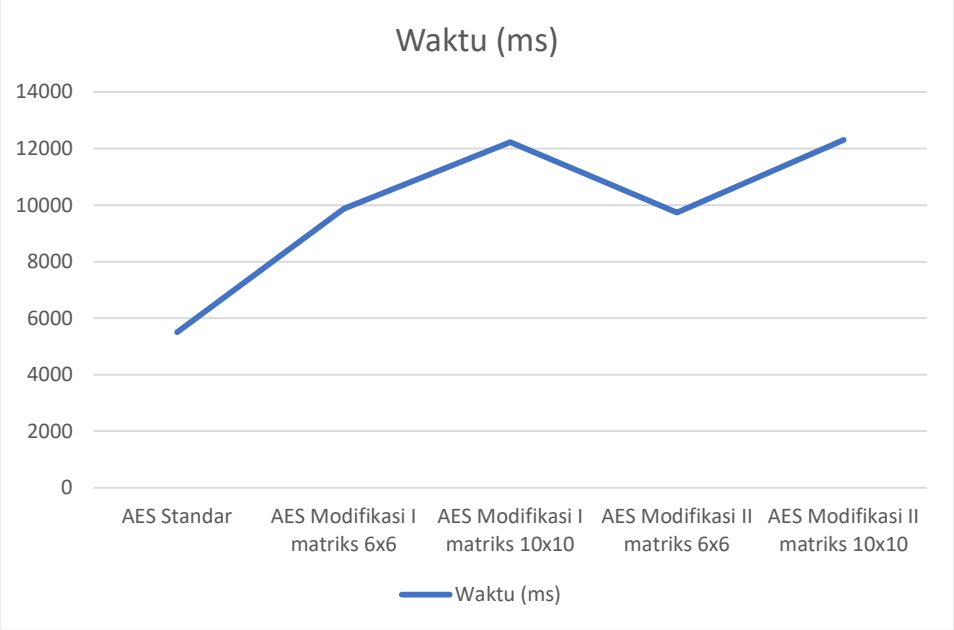
Grafik 4.2 Waktu Enkripsi



Hasil dari pengujian pada Grafik 4.2 menunjukkan hasil dengan rata-rata eksekusi waktu enkripsi tercepat pada algoritma AES standar yaitu 3391,2/detik yang berarti sesuai pada point 2 pada subbab 3.4.3 bahwa algoritma AES memiliki waktu eksekusi yang lebih baik. Perbedaan waktu eksekusi yang didapatkan antar algoritma dikarenakan adanya penambahan proses subtitusi *polybius* untuk algoritma modifikasi

sehingga mempengaruhi waktu eksekusi pada proses enkripsi. Kompleksitas algoritma berbanding lurus dengan proses waktu komputasi.

Grafik 4.3 Waktu Dekripsi



Hasil dari pengujian pada Grafik 4.3 menunjukkan hasil dengan rata-rata waktu proses dekripsi pada modifikasi AES pada plainteks dengan matriks 10x10 yaitu 12301/detik menunjukkan waktu eksekusi lebih lama sehingga sesuai pada point 1 subbab 3.4.3 bahwa algoritma modifikasi memiliki tingkat keamanan yang baik karena berpengaruh pada waktu yang dibutuhkan kriptoanalis dalam memecahkan kunci sesuai pada sub bab 2.7.3. Secara keseluruhan untuk setiap algoritma dapat dikatakan baik dari segi performa waktu karena hasil performa waktu enkripsi lebih rendah dibandingkan waktu proses dekripsi untuk masing-masing algoritma.

4.2.5 Pengujian Entropi

Hasil pengujian dapat dilakukan analisa tingkat probabilitas kemungkinan karakter yang digunakan pada proses pengamanan data berdasarkan tabel pada sub bab 3.4.4.

Tabel 4.9 Hasil pengujian entropi

No.	Algoritma	Rata-rata
1.	AES Standar	6.582,8
2.	AES Modifikasi I pada plain matriks 6x6	6.575,2
3.	AES Modifikasi I pada plain matriks 10x10	6.642,4
4.	AES Modifikasi II pada plainkey matriks 6x6	6.677,6
5.	AES Modifikasi II pada plainkey matriks 10x10	6.492,8

Berdasarkan tabel 4.9 dapat disimpulkan bahwa AES modifikasi II pada plainkey untuk ukuran matriks 6x6 memiliki nilai entropi tertinggi dibandingkan algoritma lainnya sehingga dapat dikatakan lebih aman dibandingkan algoritma lain dikarenakan penggunaan kunci lebih mendekati 8 bit atau 1 karakter .